Explaining the Link Between Causal Reasoning and Expert Behavior

William R. Swartout and Stephen W. Smoliar

USC/Information Sciences Institute

ABSTRACT: Causal reasoning can be a powerful tool, but expert diagnosticians don't seem to use it extensively in everyday practice. Yet, being able to provide the causal rationale that underlies a diagnosis or other medical decision seems to be critical in providing satisfying explanations and justifications of that decision. Thus, expert systems are presented with a paradox. It appears that they should reason non-causally in most circumstances, but still have access to the causal rationale behind their decisions for providing explanations. In this paper, we present a paradigm for expert system construction that provides that capability. In our approach, causal reasoning that is performed while the expert system is being designed does not appear in the expert system itself. But because the design process is recorded in a machine readable form, explanation routines have access to that causal reasoning and thus can justify an expert system's behavior with a causal argument. We present three increasingly sophisticated frameworks that embody this approach, XPLAIN and two versions of the Explainable Expert Systems framework.

1. Introduction

There is a paradox in causal reasoning. It can be a powerful tool in performing a difficult diagnosis [Patil 81] and is frequently used in explaining *why* a particular diagnosis is correct, but diagnosticians in a variety of domains don't seem to use it very much in performing routine diagnoses [Johnson and Moen 87]. Why does this occur?

Before this question can be confronted, it is important to recognize that causal reasoning is essentially an *abstraction* of some (usually complex) body of declarative knowledge. In the case of medicine, such knowledge will include domains such as physiology and metabolic processes. The need for abstraction of such knowledge arises from at least two reasons:

1. The amount of knowledge is too great to be systematically searched in the course of "practical" problem solving.

2. The level of the knowledge is too detailed. Solving even the most elementary problem may involve piecing together an unwieldy number of "basic facts" before one can draw a conclusion.

Reasoning based on causal relationships is a form of abstraction which makes such complexity more manageable. We may now reformulate our question by saying: why is this abstraction often rejected?

One reason is that medical knowledge is not always so easily abstracted. In many situations that knowledge may be too incomplete to admit of abstraction. In others, the knowledge may be thorough but so complex in its thoroughness that insights regarding *how* that knowledge may be abstracted are still lacking.

The other reason is that there are alternative abstractions. A major alternative in medicine, as well as other disciplines, is to abstract from the memory of past problem solving experiences, either personal or acquired from knowledge of case histories. Another is the compilation of problem solving knowledge based on either first principles or causal relations into easily recalled procedures, such as "rules of thumb." Such procedures may be highly reliable, but the practitioner may not be able to account for why they work.

Part of the appeal of current expert systems technology is that it facilitates the translation of such "rules of thumb" into a working computer program. What these systems lack, however, is any basis for justification for those rules; and while we may be willing to accept vague and informal justification from a human expert we know well enough to trust, we are more reluctant to put such trust in a software expert and require better justification to instill confidence in any conclusions such a system might draw.

A useful "middle ground" for such justification is provided by the abstraction of causal reasoning. For example, compare Figure 1-1 with Figure 1-2. The explanation in Figure 1-1 illustrates the behavior of an expert system based entirely on "rules of thumb." This system is a digitalis therapy advisor [Swartout 77], and in this example it is trying to determine whether a patient is abnormally sensitive to digitalis. When the user asks why it wants to know some data, it responds with a brief account of the rules of thumb it is trying to use. This explanation lacks any reference to the causal relations on which the rules are based. Although those relations were known to the system designer, it was not necessary to represent them for the system to *perform* in an expert fashion, hence they were not represented and are unavailable for explanation. Contrast this with the account given in Figure 1-2. The same system has now been enhanced with knowledge of causal relationships which allow it to account for its behavior in terms which are much closer to the underlying medical knowledge.

Such causal reasoning can thus be extremely valuable in enhancing the ability of an expert system to explain its own behavior. How can such causal knowledge be integrated with the "rules of thumb" under which most expert systems tend to operate? This is the question we have been exploring.

WHAT IS THE LEVEL
OF SERUM CALCIUM?

ENTER NUMERICAL VALUE ==> why?
The user types "why?" indicating that he questions why he is being asked to supply the value of serum calcium.

MY TOP GOAL IS TO BEGIN THERAPY.
ONE STEP IN DOING THAT IS TO CHECK
SENSITIVITIES. I AM NOW TRYING TO
CHECK SENSITIVITY DUE TO CALCIUM.

Figure 1-1: A Limited Justification for a Question

In the remainder of this paper, we will first describe how the XPLAIN framework employed causal relations in the process of designing a routine for dealing with drug sensitivities. We will then describe how the Explainable Expert Systems (EES) project has built on those results and allowed us to capture the design of an expert system in a more principled fashion. Version I of EES introduced an explicit representation of the terminology of the problem domain. Version II is concerned with capturing a more explicit representation of the relationship between problem solving knowledge and the underlying facts of the domain.

2. The XPLAIN framework

The XPLAIN framework [Swartout 83] and its successor, the EES project at ISI have been concerned with creating a framework for expert system development that records the reasoning that underlies the design of an expert system, so that better explanations can be provided. In our approach, domain experts and system builders collaborate to construct a high level representation of knowledge of the domain that explicitly separates different kinds of knowledge such as knowledge of how the domain works (of which causal knowledge may be a part), problem solving knowledge and knowledge of terminology. An automatic programmer is then used to derive performance-level rules or methods of the sort found in many expert systems from this abstract representation of knowledge.

Please enter the value of serum calcium: why?

The system is anticipating digitalis toxicity. Increased serum calcium causes increased automaticity, which may cause a change to ventricular fibrillation. Increased digitalis also causes increased automaticity. Thus, if the system observes increased serum calcium, it reduces the dose of digitalis due to increased serum calcium.

Please enter the value of serum calcium: 9

Please enter the value of serum potassium: why?

(The system produces a shortened explanation, reflecting the fact that it has already explained several of the causal relationships in the previous explanation. Also, since the system remembers that it has already told the user about serum calcium, and because it knows that the same plan was used to generate the code for both serum potassium and serum calcium, it suggests the analogy between the two here.)

The system is anticipating digitalis toxicity. Decreased serum potassium also causes increased automaticity. Thus, (as with increased serum calcium) if the system observes decreased serum potassium, it reduces the dose of digitalis due to decreased serum potassium.

Please enter the value of serum potassium: 3.7

Figure 1-2: An Explanation of Why Serum Calcium and Potassium are Checked Produced by XPLAIN

The derivation process is recorded in a machine-readable form, and that recorded trace is used by explanation routines to provide explanations that reflect not only the system's performance level knowledge but also the causal knowledge that it is derived from.

To determine what kinds of knowledge structures were important to model and what kinds of compilation processes we wanted to capture, we began by determining what kinds of explanations expert systems needed to offer. Using protocols and our own experience as expert system builders and users, we identified approximately a dozen different classes of useful explanations (see [Swartout 86]). We used those results to determine the kinds of knowledge structures and compilation processes to model. In this paper, we will focus on three types of questions, which are:

1. *justifications*--questions about the appropriateness of the system's actions
2. questions about the *terminology* that the system uses
3. questions about the *intent* behind the system's goals, that is, what it means to achieve a goal

Based on these results, we also identified several different kinds of knowledge involved in the creation of expert systems. In our approach to expert system construction, these different kinds of knowledge are represented separately and explicitly and then combined together by an automatic program writer to create a working expert system. The first two kinds of knowledge we identified were:

- **domain descriptive knowledge** is the knowledge that describes how the domain works. In a medical domain it is basically physiological knowledge, including knowledge of physiological parameters, diseases, possible interventions and causal relationships among them. This is typically the sort of knowledge that one finds in textbooks. What is missing from domain descriptive knowledge is the "how to" knowledge, which is our second category of knowledge.
- **problem solving knowledge** supplies knowledge about how tasks (called *goals* in our system) can be accomplished. This is where knowledge about how to perform a diagnosis or how to administer a drug belongs. In our representation, problem solving knowledge is organized into plans. Plans have *capability descriptions* which describe what goals they can achieve. Each plan also has a *method* which is a sequence of substeps (which may themselves include subgoals) for accomplishing the goal. Capability descriptions are patterns and may include variables that are bound when the capability description is matched against a goal to be achieved.

Our first experiment with this approach was the XPLAIN framework. It provided explicit representations for domain descriptive knowledge and problem solving knowledge. We used digitalis therapy as a testbed domain in developing XPLAIN. We will use an example from this domain to illustrate how the program writer worked and the kinds of knowledge that were represented. For this domain, the descriptive domain knowledge included causal relations between physiological states and characterizations of those states such as:

Increased digitalis causes increased automaticity.

Decreased serum potassium causes increased automaticity.

Increased serum calcium causes increased automaticity.

Increased automaticity may cause ventricular fibrillation.

Ventricular fibrillation is a dangerous condition.

Decreased serum potassium is an observable deviation.

Increased serum calcium is an observable deviation.

The problem solving knowledge consisted of plans (called "domain principles" in XPLAIN) for various tasks such as assessing the patient's state, gathering information about the patient and compensating the drug dose for sensitivities. As described above, these plans contained capability descriptions[1] and methods. In XPLAIN, plans also had a third component, called a *domain rationale*, which was a pattern that was matched against the domain descriptive model when the plan was instantiated. Variables in the domain rationale could appear in the steps of the method and when the steps of the plan were instantiated, the variables were replaced by their bound values.

To illustrate how this worked, consider the plan concerned with the problem of adjusting the dosage for patients who might be abnormally sensitive to digitalis. It expressed the common sense notion that if a patient had some condition that might interact with the drug in a dangerous way, then the drug dosage should be reduced. In paraphrased form, the plan was represented as[2]:

`Capability-description:` anticipate *drug* toxicity

`Domain-rationale:`
An *observable deviation* that causes a *dangerous condition* that is also caused by the *drug*.

`Method:` If the *observable deviation* exists in the patient, then reduce the *drug* dose

This plan had a capability description that stated that it could "anticipate drug toxicity"[3], its domain rationale was a pattern matched against the domain decriptive model to find those cases where some observable deviation caused something dangerous to happen that was also caused by the drug being administered. The plan's method consisted of a single conditional step that stated that if one of the observable deviations mentioned in the domain rationale existed, then the patient's dosage should be reduced. When this plan was instantiated, there were two matches for the domain rationale, one for increased serum calcium interacting with digitalis to lead to ventricular fibrillation and the other for decreased serum potassium. The program writer instantiated the plan's method twice, once for each match. The program writer also reasoned about the what should be done if multiple sensitivities occurred simultaneously (see [Swartout 81] for a description of that reasoning). This entire process was recorded so that it could later be used in giving much richer explanations that reflected the causal underpinnings that the expert system was based on, as shown in Figure 1-2. The critical difference between that explanation and the one in Figure 1-1 are the second and third sentences of the first explanation which provide a causal reason for checking serum calcium. This explanation was produced by paraphasing the causal relations that matched the domain rationale of the plan used to generate this code for checking serum calcium.

3. EES version I

While XPLAIN was capable of offering better explanations, particularly in the first category above of justifications, there were some aspects of its design that troubled us. A major problem was that the domain rationale appeared to be unmotivated in the sense that it was difficult to state precisely what role it played during system creation. Eventually, we realized that the domain rationale was actually providing an implicit definition of terminology. In the example above, the domain rationale was defining what it meant for something to be a "sensitivity". We further realized that it was inappropriate to represent terminology as part of problem solving knowledge, but instead it should have a separate representation. Such a representation would also allow us to answer questions about terminology, our second question category above. In building the first version of EES, we added *terminology* as another kind of knowledge to our framework. This explicit terminology provided us with the building blocks that were used for representing facts as part of domain descriptive knowledge and goals and methods as part of problem solving knowledge.

3.1. Adding Terminology

In most expert systems, terms and their definitions are understood by the system builder, but the terms are not explicitly defined within the system itself. Instead, the terms used by the system implicitly acquire a definition based on how other knowledge sources in the system react to them and the operational mechanisms for recognizing instances of those terms. This can lead to problems both in explanation and maintenance of an expert system. We wanted to provide an explicit and independent definition for terminology, which we defined as:

- **terminology** is knowledge of domain concepts and relationships that forms the language that knowledge sources within an expert system use to communicate.

To illustrate the problem of implicit terminology briefly, suppose we define a *very* simple rule for recognizing fever:

If patient's temperature > 100
then conclude fever.

Arguably, this rule could be considered a definition of what fever is, that is, a temperature greater than 100 degrees. An explanation routine could display the rule whenever a user wanted to know how fever was defined. In fact, that would confuse an operational means for recognizing fever with a definition for it. To see that, consider what might happen if we put our little "expert system" out in the field. We might find that we obtain many false positive results because some people drink hot coffee before taking their temperature. Of course, we can easily fix that by modifying the rule:

If patient's temperature > 100
and patient has not recently drunk coffee
then conclude fever.

Unfortunately, if we now display this rule as a definition for what fever means, the definition of fever would appear to have something to do with whether or not coffee has been consumed. The point is that an explicit definition for terminology is needed that is separate from the operational means for recognizing when some condition holds.

To provide an explicit representation for terminology, we have been using a knowledge representation system based on the ideas pioneered in KL-ONE [Brachman 78]. Our representation is based on concepts (which correspond to terms) and attributes arranged in a generalization hierarchy. As new terms are introduced, their position in this hierarchy is determined by an automatic classification facility. Since plan capabilities and goals are represented as concepts in this formalism, the generalization hierarchy of terms induces a generalization hierarchy of plans.

Taking this approach, it becomes clear that XPLAIN's domain rationale was a poor mechanism for dealing with terminology, first because it doesn't give an explicit definition for a term, and second because it confounds knowledge of terminology with problem solving knowledge. Knowledge of terminology should be shared across problem solving knowledge, not embedded as part of it. For example, in the context of the digitalis advisor, the pattern in the domain rationale of the plan for dealing with digitalis sensitivities should be removed from that plan and placed in the terminological base as a definition for the term "sensitivity". That provides a better representation for terminology, but leaves open the question of exactly how that terminology gets used during the program writing process. At least a partial answer to that question came from

[1] Somewhat misleadingly, capability descriptions were called "goals" in XPLAIN.

[2] Pattern variables are in italics.

[3] We now feel that it would be more appropriate to call this capabiltiy "compensate for drug sensitivities"

addressing another limitation of XPLAIN, and providing EES version I with a program writer capable of reformulating goals.

3.2. Adding Reformulations

The power of XPLAIN's program writer was quite limited. Although XPLAIN allowed a system builder to express the capabilities of a plan as a pattern that included variables, if a goal was posted and no matching plan could be found for it, the program writer was stuck. It had no capability to reformulate such a goal into a goal or set of goals for which plans could be found. We decided to add such a capability because we anticipated that in would provide several benefits. Maintenance and initial system construction would be easier because the program writer would be able to bridge larger gaps between plans and goals, and knowledge would be re-usable in a larger range of situations. As we will describe below, an additional benefit was that the reformulation capability together with an explicit representation of terminology allowed us to give a more explicit account of some of the implicit operations in XPLAIN.

We identified several different kinds of reformulations (see [Neches, et al. 85] for a detailed discussion). The reformulation we will focus on here is a special case of *reformulation into cases*, that is, reformulating a goal of an action to be performed over a set of objects into a set of goals where the action is performed on individual elements of the original set of objects. This is a kind of reformulation that takes place frequently (but implicitly) in expert systems.

For example, in many diagnostic systems, a problem that arises is to determine how likely it is that a patient has some disease based on its signs and symptoms. In conventional expert systems, that goal is usually not explicitly represented in the system because the system designer mentally reformulates it while constructing the system. What does appear is the result of the reformulation: a set of goals that inquire about each of the symptoms individually and a combining function that deals with the problem of how to combine the individual assessments of signs and symptoms into an appropriate overall assessment for the disease. EES allowed us to represent the original goal, the reformulation, and the result of the reformulation explicitly.

We also realized that XPLAIN had been implicitly reformulating goals. For example, in the plan for dealing with drug sensitivities that we described above, the capability description of the plan stated that the plan could deal with compensating for all sensitivities, but the method of the plan could only compensate for individual sensitivities. Clearly, some implicit reformulation was taking place.

If we wanted to re-implement the digitalis advisor using EES[4], we would model the reformulation more explicitly. The problem solving knowledge would consist of a plan whose capability description stated that it could "compensate for *a* drug sensitivity". The plan's method would be similar to the method of the plan in XPLAIN. The term "drug sensitivity" would be defined explicitly as an observable deviation that caused something dangerous that was also caused by the drug. When the goal of compensating for digitalis sensitivities was posted, the program writer would find that no plans existed for dealing with sets of sensitivities, so it would be necessary to reformulate the goal into a set of goals over individual sensitivities. Using the definition of sensitivity together with the domain descriptive knowledge above, the programmer would find that increased serum calcium and decreased serum potassium were individual sensitivities, so the original goal of compensating for digitalis sensitivities would be reformulated into two goals:

```
compensate for increased serum calcium
compensate for decreased serum potassium
```

These goals could then be implemented by the plan for an individual sensitivity.

[4]We did not actually carry out this re-implementation in EES version I. Certain limitations on the expressive power of NIKL would have made it difficult to represent terms involving transitive relations, like sensitivity. We are currently in the process of carrying out the re-implementation of portions of the digitalis advisor in EES version II, which uses a more expressive knowledge representation.

We feel that this approach captures more explicitly the program writing process that was taking place in XPLAIN, and it allows us to capture the knowledge needed to answer additional kinds of questions, such as questions about terminology. We also feel that this gives a good account for one of the ways causal knowledge is compiled into expert systems:

Causal knowledge, together with knowledge of terminology, is used during goal reformulation.

In the remainder of the paper we will describe our efforts to capture the knowledge needed to answer the third type of question listed above, questions about the intent of goals. This investigation led us to discover another way in which causal knowledge can be compiled into expert systems.

4. EES version II

While version I of EES allowed us to represent terminology and model reformulation, there were two open issues that we wanted to address. First, we wanted to be able to represent the *intent* behind a goal. For example, it was not possible to answer the question: "What does it mean to administer digitalis?" Problem solving knowledge of *how* to give digitalis could be retrieved, but it was not represented anywhere that the problem of digitalis adminsitration was a problem of producing satisfactory therapeutic results subject to the constraint of avoiding (or minimizing) toxic effects.

For another example, consider an expert system we built in EES version I for diagnosing space telemetry systems. This system had several methods for diagnosis, which we have hand paraphrased in Figure 4-1. These display what the system does in performing a diagnosis, but it takes considerable deductive effort on the part of the user to figure out what a diagnosis amounts to. What we want is an explicit representation of the intent behind the goal in this domain that would allow us to answer: "To diagnose a decomposable system means to find a primitive subcomponent of the system that is faulty." In general, expert systems lack any such specification of what their goals mean. The problem is directly analogous to the problem of implicit terminology cited above. Goals acquire their meaning based solely on the methods that claim to implement them. What we want is a separate definition for goals that a user could use as an independent criterion in deciding whether or the systems goals met his own and whether its methods were appropriate for achieving those goals.

The second issue we wanted to address was to better understand the source of problem solving knowledge. While both XPLAIN and version I of EES allowed a system builder to represent problem solving knowledge at a more abstract level than is possible in most expert system frameworks, it was clear that even those methods were compiled from some still more basic representations of knowledge. By understanding the "roots" of problem solving knowledge, we could provide better explanations of how the plans worked.

Below, we will describe the approach we have adopted, which is to represent goal intent in terms of a small number of primitive actions and then to mechanically derive plans for achieving those goals by transforming definitions and axioms in the domain descriptive knowledge into plans. We have been exploring this approach in the context of a non-medical domain, namely diagnosis of digital circuits, but we expect the approach to carry over into medical domains and are currently starting to re-implement portions of the digitalis advisor using this approach. Most of the examples in the remainder of the paper will drawn from our work with digital circuits, but we will outline some of the issues the medical domain is raising below.

```
To diagnose a decomposable system,
  If there is a fault in the system,
   then locate the cause of the fault
    within the system

To diagnose a primitive system,
  If the system is faulty,
   then conclude it is the diagnosis

To locate the cause of a fault within a
  system which is loosely-coupled,
   Diagnose the subcomponents of the system

To locate the cause of a fault within a
  system which is tightly-coupled,
   Locate the cause of the fault along the
    signal-path beginning at the system-
     input and ending at the system-output.

To locate the cause of a fault beginning at
  system1 and ending at system2,
   If system1 is faulty
    then diagnose system1
     else locate the cause of the fault
      along the signal-path beginning at
       the system that system1 outputs to
        and ending at system2.
```

Figure 4-1: Methods as an Inadequate Explanation of the Goal of Diagnosis

4.1. Capturing Intent

To capture intent, we begin by defining a set of primitive actions. These are actions that are assumed to be readily understood by users. All higher level goals are ultimately defined in terms of these primitive actions. So far we have identified four primitive actions:

1. ***determine-whether:*** establishes the truth of a given assertion
2. ***find:*** finds an object that matches a given description
3. ***achieve:*** achieves a particular state
4. ***avoid:*** the counterpart of achieve, it insures that a particular state does *not* occur.

We believe that this set of actions will grow somewhat as we gain more experience with this approach. We used the first two actions extensively in our system for diagnosing digital circuits. Interestingly, those actions correspond to the kinds of problems analyzed by Polya [Polya 71]: problems to prove and problems to find. We have found the last two actions to be useful in our analysis of the digitalis therapy advisor, since much of digitalis therapy is concerned with achieving a therapeutic effect while avoiding toxicity. Given these primitive actions, capturing the intent behind a domain level goal then involves linking that goal to its definition in terms of primitive actions. Thus, we would define the goal:

"diagnose decomposable digital system *s*"

as the problem:

"*finding* a primitive system *p* such that *p* is a subcomponent of *s* and *p* is faulty"

4.2. The Sources of Problem Solving Knowledge

The primitive actions allow us to capture the intent behind goals, but the issue still remains of how to implement the plans for realizing those goals. In EES version II, plans enter the knowledge base in two ways.

Mechanically derived plans

The first way is that plans for performing primitive actions are mechanically derived by performing transformations on the assertions and definitions in the domain descriptive knowledge base. For a very simple example, if the knowledge base contains the assertion that "A exists if and only if B exists," then it is possible to derive a plan that determines whether B exists by checking for the presence of A. Since the implication is two-way, it is also possible to derive another plan for checking for the existence of A by checking for B[5]. Considerably more complex examples can be handled. In constructing the digital circuit diagnoser, the domain descriptive model was a functional description of the interconnections within the circuit and the functional behavior of the devices in the circuit. Given that description, it was possible to mechanically derive a set of procedures for *finding* the expected signal value along any connector in the circuit, given a particular set of input values.

An interesting observation is emerging from our initial work in the digitalis domain, which is that different kinds of primitive actions seem to involve transformations over different kinds of domain descriptive knowledge. The transformations for deriving plans for performing *find* actions involve sets and instances, *determine-whether* involves implications and types, and *achieve* and *avoid* involve states, state transitions, and causality. Thus, we are finding that another way that mechanistic descriptions or causal relations can be compiled into an expert system is by:

Direct translation into methods for performing primitive actions.

Weak methods

The other way that plans can enter the knowledge base is by being entered by hand. That may be obvious, but there is one category of such plans that deserves special attention. We call these plans *weak methods*. These are very general plans, not specific to any domain, that provide a means for performing achieving the primitive actions in some very general circumstances. For example, one of our weak methods involving *determine-whether* states that the problem of determining the truth of a conjunction of two assertions can be performed by determining the truth of each assertion in turn and combining the results in the obvious way. Another weak method for finding an object that matches a description is the classic *generate-and-test*.

Our view of weak methods differs from previous ones, such as [Laird 83]. Such views regard the weak methods themselves as primitive elements, and problem solving as the application of operators within a problem space. The alternative view which we propose is to regard problem solving in terms a foundation of primitive actions which are the primitive elements. Weak methods, then, are concerned with providing general ways for achieving those actions.

Our experience with the second version of EES is still in the early stages. Nevertheless, it appears to provide some of the additional knowledge structures we need to answer our third category of question, i.e. questions about intent.

5. Status

XPLAIN and EES version I have been implemented and used to construct demonstration sized expert systems in medical and non-medical domains. We have tested EES version II on the problem of locating a faulty component in a digital circuit, and are now engaged

[5] Of course, care must be taken in interpreting such plans to avoid circular reasoning chains.

in using it to re-implement portions of the digitalis advisor. An explanation facility was implemented for XPLAIN and we are currently constructing one for EES.

6. Summary

We have argued that even though experts may not use causal reasoning, such reasoning is a useful abstraction that underlies many of their decisions. We also asserted that causal reasoning must be accessible to provide good explanations and therefore warrants explicit representation in an expert system. We described three systems that have pursued the explicit representation of causal knowledge, the application of that knowledge to problem solving, and the use of that knowledge to explain problem solving behavior.

ACKNOWLEDGEMENTS

The research described here was supported under DARPA Grant #MDA 903-81-C-0335 and National Institutes of Health Grant #1 P01 LM 03374-01 from the National Library of Medicine.

References

[Brachman 78] Brachman, R., *A Structural Paradigm for Representing Knowledge*, Bolt, Beranek & Newman, Inc., Technical Report, 1978.

[Johnson and Moen 87] Johnson, P. J., and Moen, J. B., *Garden Path Errors in Diagnostic Reasoning*, Springer-Verlag, 1987. (to appear)

[Laird 83] Laird, J. and Newell, A., *A Universal Weak Method*, Carnegie-Mellon University Department of Computer Science, Pittsburgh, PA, Technical Report CMU-CS-83-141, June 1983.

[Neches, et al. 85] Neches, R., W. Swartout, J. Moore, "Enhanced Maintenance and Explanation of Expert Systems through Explicit Models of Their Development," *Transactions On Software Engineering*, November 1985. Revised version of article in Proceedings of the IEEE Workshop on Principles of Knowledge-Based Systems, December, 1984

[Patil 81] Patil, R., *Causal representation of patient illness for electrolyte and acid-base diagnosis*, Ph.D. thesis, Massachusetts Institute of Technology, 1981. (available as MIT/LCS/TR-267)

[Polya 71] Polya, G., *How To Solve It: A New Aspect of Mathematical Method,* Princeton University Press, Princeton, NJ, 1971. Second edition.

[Swartout 77] Swartout, W.R., *A Digitalis Therapy Advisor with Explanations*, Massachusetts Institute Technology, Technical Report Laboratory for Computer Science TR-176, February 1977.

[Swartout 81] Swartout, W., *Producing explanations and justifications of expert consulting systems*, MIT, Technical Report 251, 1981.

[Swartout 83] Swartout, W., "XPLAIN: A system for creating and explaining expert consulting systems," *Artificial Intelligence* 21, (3), September 1983, 285-325. Also available as ISI/RS-83-4

[Swartout 86] Swartout, W., "Knowledge Needed for Expert System Explanation," *Future Computing Systems* 1, (2), 1986.